Open in Colab

```
!pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.34.1-py3-none-any.whl (7.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 59.4 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.4)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.0/302.0 kB 41.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers<0.15,>=0.14 (from transformers)
  Downloading tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 106.4 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 88.2 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4-
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 295.0/295.0 kB 40.1 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.22)
Installing collected packages: safetensors, huggingface-hub, tokenizers, transformers
Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0 tokenizers-0.14.1 transformers-4.34.1
```

```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.14.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes==0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.34.0
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.0)
Requirement already satisfied: tensorboard<2.15,>=2.14 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: keras<2.15,>=2.14.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.41.
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflo
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (3.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.15,>=2.14->tensorflow) (3.
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboar
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<1.1,>=0.5-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboar
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.1
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.15
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.15,>=2.
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oa
from transformers import BertTokenizer, TFBertModel, TFBertForSequenceClassification, RobertaTokenizer, TFRobertaForSequenceClassification,Au
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras import layers
import sklearn
import pandas as pd
import io
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

```
from google.colab import files
uploaded_train = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving train.csv to train.csv
```

```
from google.colab import files
uploaded_test = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test.csv to test.csv
```

EDA

```
# Loading the dataset
df = pd.read_csv(io.BytesIO(uploaded_train['train.csv']))
df_test = pd.read_csv(io.BytesIO(uploaded_test['test.csv']))
```

```
# Printing the first 5 rows of the DataFrame
df.head()
```

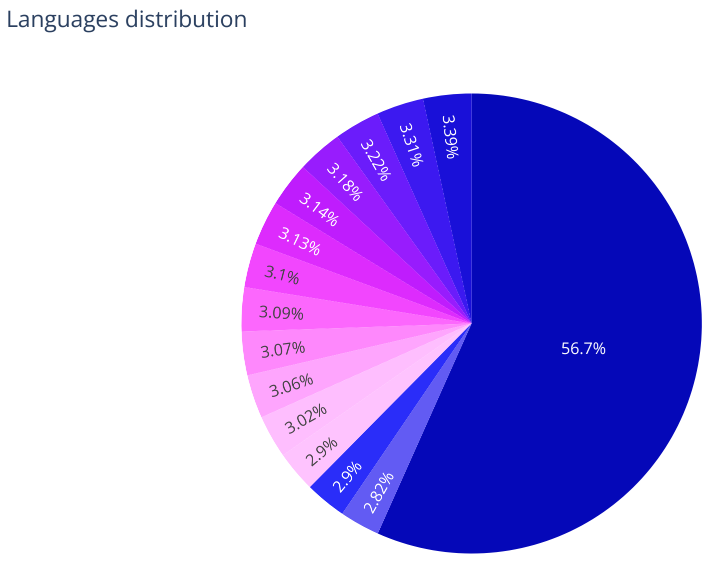
	id	premise	hypothesis	lang_abv	language	label
0	5130fd2cb5	and these comments were considered in formulat...	The rules developed in the interim were put to...	en	English	0
1	5b72532a0b	These are issues that we wrestle with in pract...	Practice groups are not permitted to work on t...	en	English	2
2	3931fbe82a	Des petites choses comme celles-là font une di	J'essayais d'accomplir quelque chose.	fr	French	0

```
df_test.head()
```

	id	premise	hypothesis	lang_abv	language
0	c6d58c3f69	بکس، کیسی، راہیل، یسعیاہ، کیلی، کیلی، اور کولم...	کیسی کے لئے کوئی یادگار نہیں ہوگا, کولمین ہائی...	ur	Urdu
1	cefcc82292	هذا هو ما تم نصحنا به.	عندما يتم إخبارهم بما يجب عليهم فعله ، فشلت ال...	ar	Arabic
2	e98005252c	et cela est en grande partie dû au fait que le...	Les mères se droguent.	fr	French
		与城市及其他公民及社区组织	IMA 与其他组织合作，因为		

```
import plotly.express as px

labels, frequencies = np.unique(df.language.values, return_counts = True)

fig = px.pie(values=frequencies,
             names=labels,
             title='Languages distribution',
             color_discrete_sequence=px.colors.sequential.Plotly3)
```

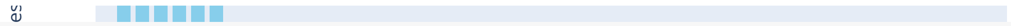
```
fig.show()
```



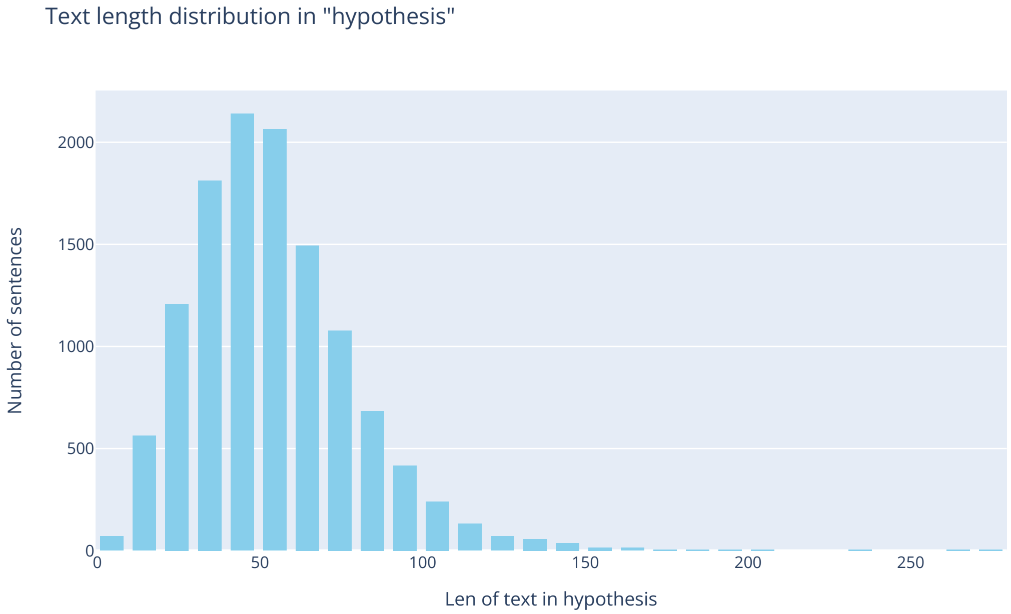
Based on the above chart, we can clearly see that there are a total of 15 languages of which 56.7% is from English language and the remaining 42.3% is divided among rest of the 14 languages.

```
import plotly.graph_objects as go

df['text_length'] = df['premise'].apply(len)

fig = go.Figure(data=[go.Histogram(x=df['text_length'],
                                   nbinsx=50,
                                   marker_color='skyblue')])
fig.update_layout(title_text='Text length distribution in "premise"', # title of plot
                  xaxis_title_text='Len of text in premise', # xaxis label
                  yaxis_title_text='Number of sentences', # yaxis label
                  bargap=0.2, # gap between bars of adjacent location coordinates
                  bargroupgap=0.1) # gap between bars of the same location coordinates
fig.show()
```

There are more than 1500 sentences with less than 100 length of text in premise

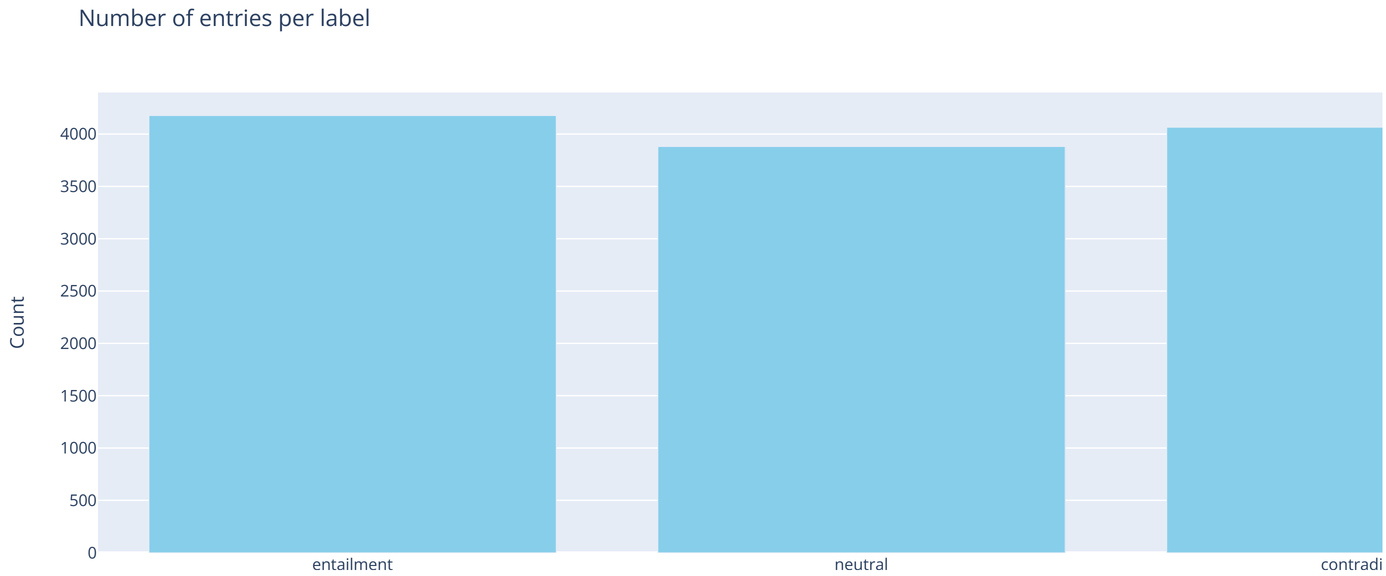


```
df['text_length'] = df['hypothesis'].apply(len)

fig = go.Figure(data=[go.Histogram(x=df['text_length'],
                                   nbinsx=50,
                                   marker_color='skyblue')])
fig.update_layout(title_text='Text length distribution in "hypothesis"', # title of plot
                  xaxis_title_text='Len of text in hypothesis', # xaxis label
                  yaxis_title_text='Number of sentences', # yaxis label
                  bargap=0.2, # gap between bars of adjacent location coordinates
                  bargroupgap=0.1) # gap between bars of the same location coordinates
fig.show()
```



The above graph shows that there are more than 2000 sentences with Length of text 50 in hypothesis

```
label_count = df['label'].value_counts().sort_index()
label_names = ['entailment', 'neutral', 'contradiction']
label_count.index = label_names

fig = go.Figure([go.Bar(x=label_names, y=label_count, marker_color='skyblue')])

fig.update_layout(title_text='Number of entries per label', # title of plot
                  xaxis_title_text='Label', # xaxis label
                  yaxis_title_text='Count', # yaxis label
                  )
fig.show()
```



Based on the above chart, we can see clearly that the dataset is well balanced

RoBERTa-based Sequence Classification for Natural Language Inference

```
# Preparing the dataset
X = df[['premise', 'hypothesis']].values
y = df['label'].values
```

```
# Initializing the RoBERTa tokenizer
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
```

```
Downloading (…)olve/main/vocab.json: 100%          899k/899k [00:00<00:00, 21.5MB/s]
Downloading (…)olve/main/merges.txt: 100%          456k/456k [00:00<00:00, 1.00MB/s]
Downloading (…)/main/tokenizer.json: 100%          1.36M/1.36M [00:00<00:00, 2.97MB/s]
Downloading (…)lve/main/config.json: 100%          481/481 [00:00<00:00, 40.4kB/s]
```

```
# Splitting the dataset into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Tokenizing and encoding the data
def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs
```

```
train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)

# Preparing the input data
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)

# Here we are creating a custom callback for early stopping
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)
```

```
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the 'longest_first' truncatior
```

```
# Initializing the RoBERTa model
model = TFRobertaForSequenceClassification.from_pretrained('roberta-base', num_labels=3)
```

```
Downloading model.safetensors: 100%                    499M/499M [00:01<00:00, 469MB/s]

Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFRobertaForSequenceClassification: ['roberta.embeddi
- This IS expected if you are initializing TFRobertaForSequenceClassification from a PyTorch model trained on another task or with anoth
- This IS NOT expected if you are initializing TFRobertaForSequenceClassification from a PyTorch model that you expect to be exactly ide
Some weights or buffers of the TF 2.0 model TFRobertaForSequenceClassification were not initialized from the PyTorch model and are newly
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# Compiling the model
optimizer = keras.optimizers.Adam(learning_rate=2e-5)
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

```
# Training the model with early stopping
history = model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping
```

```
Epoch 1/20
303/303 [==============================] - 109s 237ms/step - loss: 1.0230 - accuracy: 0.4322 - val_loss: 0.8502 - val_accuracy: 0.5875
Epoch 2/20
303/303 [==============================] - 65s 214ms/step - loss: 0.7804 - accuracy: 0.6156 - val_loss: 0.7830 - val_accuracy: 0.6304
```

```
Epoch 3/20
303/303 [==============================] - 64s 211ms/step - loss: 0.6352 - accuracy: 0.6956 - val_loss: 0.7890 - val_accuracy: 0.6304
Epoch 4/20
303/303 [==============================] - 63s 209ms/step - loss: 0.5383 - accuracy: 0.7401 - val_loss: 0.8666 - val_accuracy: 0.6411
Epoch 5/20
303/303 [==============================] - 63s 210ms/step - loss: 0.4769 - accuracy: 0.7676 - val_loss: 0.9741 - val_accuracy: 0.6308
```

```
# Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
76/76 [==============================] - 6s 74ms/step - loss: 0.7830 - accuracy: 0.6304
Test Loss: 0.7829666137695312
Test Accuracy: 0.6303630471229553
```

Multilingual BERT (mBERT) Sequence Classification for Natural Language Inference with Optimization Techniques (EarlyStopping to prevent overfitting)

```
# Initializing the mBERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
```

```
Downloading (…)okenizer_config.json: 100%        29.0/29.0 [00:00<00:00, 2.69kB/s]

Downloading (…)solve/main/vocab.txt: 100%        996k/996k [00:00<00:00, 4.36MB/s]

Downloading (…)/main/tokenizer.json: 100%        1.96M/1.96M [00:00<00:00, 4.30MB/s]

Downloading (…)lve/main/config.json: 100%        625/625 [00:00<00:00, 54.1kB/s]
```

```
# Tokenizing and encoding the data
def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs
```

```
train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)

# Preparing the input data
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

```
# Compiling the model
optimizer = keras.optimizers.Adam(learning_rate=2e-5)
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

```
# Training the model with early stopping
history = model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping
```

```
Epoch 1/20
303/303 [==============================] - 105s 239ms/step - loss: 0.9327 - accuracy: 0.5549 - val_loss: 0.8264 - val_accuracy: 0.6258
Epoch 2/20
303/303 [==============================] - 66s 217ms/step - loss: 0.7013 - accuracy: 0.7078 - val_loss: 0.8204 - val_accuracy: 0.6572
Epoch 3/20
303/303 [==============================] - 65s 214ms/step - loss: 0.4904 - accuracy: 0.8098 - val_loss: 0.9284 - val_accuracy: 0.6448
Epoch 4/20
303/303 [==============================] - 64s 211ms/step - loss: 0.3146 - accuracy: 0.8852 - val_loss: 1.0922 - val_accuracy: 0.6427
Epoch 5/20
303/303 [==============================] - 64s 212ms/step - loss: 0.1947 - accuracy: 0.9322 - val_loss: 1.3186 - val_accuracy: 0.6361
```

```
# Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
76/76 [==============================] - 6s 75ms/step - loss: 0.8204 - accuracy: 0.6572
Test Loss: 0.8204382061958313
Test Accuracy: 0.6571782231330872
```

XLM-RoBERTa Sequence Classification for Natural Language Inference with Enhanced Training Techniques

```
X = df[['premise', 'hypothesis']].values
y = df['label'].values


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)


train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

```
tokenizer = AutoTokenizer.from_pretrained("xlm-roberta-base")
model = TFAutoModelForSequenceClassification.from_pretrained("xlm-roberta-base", num_labels=3)
```

```
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are ne
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs
```

```
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

```
# Compiling the model
optimizer = keras.optimizers.Adam(learning_rate=2e-5)
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

```
# Training the model with early stopping
history = model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping
```

```
Epoch 1/20
303/303 [==============================] - 109s 256ms/step - loss: 1.0991 - accuracy: 0.3598 - val_loss: 1.0799 - val_accuracy: 0.4328
Epoch 2/20
303/303 [==============================] - 67s 222ms/step - loss: 1.0731 - accuracy: 0.4027 - val_loss: 1.0534 - val_accuracy: 0.4191
Epoch 3/20
303/303 [==============================] - 66s 218ms/step - loss: 0.9997 - accuracy: 0.4827 - val_loss: 0.8586 - val_accuracy: 0.6229
Epoch 4/20
303/303 [==============================] - 66s 217ms/step - loss: 0.8174 - accuracy: 0.6406 - val_loss: 0.7303 - val_accuracy: 0.6881
Epoch 5/20
303/303 [==============================] - 65s 214ms/step - loss: 0.6261 - accuracy: 0.7445 - val_loss: 0.7623 - val_accuracy: 0.6935
Epoch 6/20
303/303 [==============================] - 65s 214ms/step - loss: 0.4605 - accuracy: 0.8194 - val_loss: 0.7627 - val_accuracy: 0.7096
Epoch 7/20
303/303 [==============================] - 65s 213ms/step - loss: 0.3210 - accuracy: 0.8818 - val_loss: 0.8960 - val_accuracy: 0.6993
Epoch 8/20
303/303 [==============================] - 65s 213ms/step - loss: 0.2242 - accuracy: 0.9214 - val_loss: 0.9652 - val_accuracy: 0.7013
Epoch 9/20
303/303 [==============================] - 65s 216ms/step - loss: 0.1586 - accuracy: 0.9461 - val_loss: 1.1213 - val_accuracy: 0.7046
```

```
# Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
76/76 [==============================] - 6s 75ms/step - loss: 0.7303 - accuracy: 0.6881
Test Loss: 0.7302846908569336
Test Accuracy: 0.6881188154220581
```

Fine-Tuning XLM-RoBERTa for Natural Language Inference with Custom Architecture and Dropout Regularization

```
X = df[['premise', 'hypothesis']].values
y = df['label'].values


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

```
tokenizer = AutoTokenizer.from_pretrained("xlm-roberta-base")

# Encoding function
def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs
```

```
def create_model():
    base_model = TFAutoModelForSequenceClassification.from_pretrained("xlm-roberta-base", num_labels=3)
    input_ids = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="input_ids")
    attention_mask = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="attention_mask")

    outputs = base_model([input_ids, attention_mask]).logits
    dropout = tf.keras.layers.Dropout(0.3)(outputs)  # adding dropout for regularization
    outputs = tf.keras.layers.Dense(3, activation='softmax')(dropout)

    model = tf.keras.Model(inputs=[input_ids, attention_mask], outputs=outputs)
    return model

model = create_model()


early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are ne
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# Compiling the model
optimizer = keras.optimizers.Adam(learning_rate=2e-5)
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

```
# Callback for early stopping
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```
history = model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/backend.py:5729: UserWarning:
```

```
"`sparse_categorical_crossentropy` received `from_logits=True`, but the `output` argument was produced by a Softmax activation and thus

303/303 [==============================] - 110s 260ms/step - loss: 1.0930 - accuracy: 0.3583 - val_loss: 0.9933 - val_accuracy: 0.5384
Epoch 2/20
303/303 [==============================] - 67s 222ms/step - loss: 0.9637 - accuracy: 0.5209 - val_loss: 0.8176 - val_accuracy: 0.6432
Epoch 3/20
303/303 [==============================] - 66s 219ms/step - loss: 0.7975 - accuracy: 0.6350 - val_loss: 0.7304 - val_accuracy: 0.6885
Epoch 4/20
303/303 [==============================] - 66s 216ms/step - loss: 0.6845 - accuracy: 0.6880 - val_loss: 0.7226 - val_accuracy: 0.6955
Epoch 5/20
303/303 [==============================] - 65s 213ms/step - loss: 0.5677 - accuracy: 0.7397 - val_loss: 0.8372 - val_accuracy: 0.6951
Epoch 6/20
303/303 [==============================] - 65s 213ms/step - loss: 0.4887 - accuracy: 0.7706 - val_loss: 0.8865 - val_accuracy: 0.7162
Epoch 7/20
303/303 [==============================] - 65s 214ms/step - loss: 0.4307 - accuracy: 0.7874 - val_loss: 1.0531 - val_accuracy: 0.6799
Epoch 8/20
303/303 [==============================] - 64s 213ms/step - loss: 0.3587 - accuracy: 0.8156 - val_loss: 1.0304 - val_accuracy: 0.7034
Epoch 9/20
303/303 [==============================] - 65s 216ms/step - loss: 0.3189 - accuracy: 0.8288 - val_loss: 1.1733 - val_accuracy: 0.7054
```

```
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
76/76 [==============================] - 6s 75ms/step - loss: 0.7226 - accuracy: 0.6955
Test Loss: 0.722647488117218
Test Accuracy: 0.6955445408821106
```

Layer-wise Fine-Tuning of XLM-RoBERTa for Natural Language Inference with Learning Rate Scheduling

```
X = df[['premise', 'hypothesis']].values
y = df['label'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

```
def create_model():

    base_model = TFAutoModelForSequenceClassification.from_pretrained("xlm-roberta-base", num_labels=3).roberta

    input_ids = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="input_ids")
    attention_mask = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="attention_mask")

    embeddings = base_model([input_ids, attention_mask])
    pooled_output = embeddings[0][:, 0, :]
    dropout = tf.keras.layers.Dropout(0.3)(pooled_output)
    outputs = tf.keras.layers.Dense(3, activation='softmax')(dropout)

    model = tf.keras.Model(inputs=[input_ids, attention_mask], outputs=outputs)
    return model


model = create_model()
```

```
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are ne
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Learning rate scheduler
lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-5 * 10**(epoch / 20))
```

```
# Compiling the model
optimizer = keras.optimizers.Adam(learning_rate=1e-5)
```

```
loss = keras.losses.SparseCategoricalCrossentropy()
metric = keras.metrics.SparseCategoricalAccuracy('accuracy')
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

```
history = model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping

# After a few epochs, here we are unfreezing the last few layers of RoBERTa and continue training.
for layer in model.layers[-5:]:
    layer.trainable = True

optimizer = keras.optimizers.Adam(learning_rate=5e-6)
model.compile(optimizer=optimizer, loss=loss, metrics=[metric])

history_finetune = model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_
```

```
Epoch 1/20
303/303 [==============================] - 110s 258ms/step - loss: 1.2769 - accuracy: 0.3861 - val_loss: 0.9335 - val_accuracy: 0.5726 -
Epoch 2/20
303/303 [==============================] - 67s 221ms/step - loss: 0.9097 - accuracy: 0.6050 - val_loss: 0.7273 - val_accuracy: 0.6943 -
Epoch 3/20
303/303 [==============================] - 66s 216ms/step - loss: 0.7216 - accuracy: 0.7102 - val_loss: 0.7276 - val_accuracy: 0.7054 -
Epoch 4/20
303/303 [==============================] - 65s 213ms/step - loss: 0.5816 - accuracy: 0.7717 - val_loss: 0.7856 - val_accuracy: 0.7133 -
Epoch 5/20
303/303 [==============================] - 65s 213ms/step - loss: 0.4508 - accuracy: 0.8278 - val_loss: 0.8211 - val_accuracy: 0.7046 -
Epoch 6/20
303/303 [==============================] - 65s 214ms/step - loss: 0.3375 - accuracy: 0.8789 - val_loss: 0.8466 - val_accuracy: 0.7100 -
Epoch 7/20
303/303 [==============================] - 65s 216ms/step - loss: 0.2575 - accuracy: 0.9143 - val_loss: 1.1301 - val_accuracy: 0.6823 -
Epoch 1/20
303/303 [==============================] - 107s 256ms/step - loss: 0.6878 - accuracy: 0.7149 - val_loss: 0.7128 - val_accuracy: 0.7026
Epoch 2/20
303/303 [==============================] - 67s 220ms/step - loss: 0.6055 - accuracy: 0.7622 - val_loss: 0.7674 - val_accuracy: 0.7104
Epoch 3/20
303/303 [==============================] - 65s 216ms/step - loss: 0.5451 - accuracy: 0.7895 - val_loss: 0.7701 - val_accuracy: 0.7100
Epoch 4/20
303/303 [==============================] - 65s 214ms/step - loss: 0.4786 - accuracy: 0.8202 - val_loss: 0.7948 - val_accuracy: 0.7162
Epoch 5/20
303/303 [==============================] - 65s 214ms/step - loss: 0.4082 - accuracy: 0.8518 - val_loss: 0.8606 - val_accuracy: 0.7170
Epoch 6/20
303/303 [==============================] - 66s 216ms/step - loss: 0.3547 - accuracy: 0.8690 - val_loss: 0.9185 - val_accuracy: 0.7137
```

```
test_loss, test_accuracy = model.evaluate(val_inputs, val_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
76/76 [==============================] - 6s 75ms/step - loss: 0.7128 - accuracy: 0.7026
Test Loss: 0.7127985954284668
Test Accuracy: 0.7025577425956726
```

Ensemble Learning for Natural Language Inference with Stacking and Averaging Strategies on XLM-RoBERTa

```
X = df[['premise', 'hypothesis']].values
y = df['label'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
tokenizer = AutoTokenizer.from_pretrained("xlm-roberta-base")

# Encoding function
def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs

train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

```
def create_model():
    base_model = TFAutoModelForSequenceClassification.from_pretrained("xlm-roberta-base", num_labels=3).roberta
    input_ids = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="input_ids")
    attention_mask = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="attention_mask")
    embeddings = base_model([input_ids, attention_mask])
    pooled_output = embeddings[0][:, 0, :]

    x = tf.keras.layers.BatchNormalization()(pooled_output)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.4)(x)
    x = tf.keras.layers.Dense(256, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.4)(x)
    outputs = tf.keras.layers.Dense(3, activation='softmax')(x)

    model = tf.keras.Model(inputs=[input_ids, attention_mask], outputs=outputs)
    return model
```

```
NUM_MODELS = 3
models = []
stacked_features_train = []
stacked_features_val = []

for i in range(NUM_MODELS):
    print(f"Training model {i+1}/{NUM_MODELS}...")
    model = create_model()
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-5 * 10**(epoch / 20))
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5)
    loss = tf.keras.losses.SparseCategoricalCrossentropy()
    metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
    model.compile(optimizer=optimizer, loss=loss, metrics=[metric])


    model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping, lr_s

    train_preds = model.predict(train_inputs)
    val_preds = model.predict(val_inputs)
    stacked_features_train.append(train_preds)
    stacked_features_val.append(val_preds)


    models.append(model)
    del model
    tf.keras.backend.clear_session()

# Stacking
stacked_features_train = np.concatenate(stacked_features_train, axis=1)
stacked_features_val = np.concatenate(stacked_features_val, axis=1)

stacker = keras.Sequential([
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(3, activation='softmax')
])
stacker.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
stacker.fit(stacked_features_train, y_train, validation_data=(stacked_features_val, y_test), epochs=50, batch_size=32)
```

```
Training model 1/3...
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
303/303 [==============================] - 109s 258ms/step - loss: 1.2638 - accuracy: 0.3395 - val_loss: 1.0933 - val_accuracy: 0.3523
Epoch 2/20
303/303 [==============================] - 67s 223ms/step - loss: 1.1594 - accuracy: 0.4001 - val_loss: 1.0007 - val_accuracy: 0.5289
Epoch 3/20
303/303 [==============================] - 66s 219ms/step - loss: 1.0102 - accuracy: 0.5396 - val_loss: 0.8323 - val_accuracy: 0.6349
Epoch 4/20
303/303 [==============================] - 65s 216ms/step - loss: 0.8445 - accuracy: 0.6446 - val_loss: 0.7379 - val_accuracy: 0.6877
Epoch 5/20
303/303 [==============================] - 65s 214ms/step - loss: 0.7304 - accuracy: 0.7099 - val_loss: 0.7736 - val_accuracy: 0.6790
Epoch 6/20
303/303 [==============================] - 65s 213ms/step - loss: 0.5972 - accuracy: 0.7762 - val_loss: 0.7836 - val_accuracy: 0.7050
Epoch 7/20
303/303 [==============================] - 65s 214ms/step - loss: 0.4841 - accuracy: 0.8259 - val_loss: 0.9580 - val_accuracy: 0.6852
Epoch 8/20
303/303 [==============================] - 65s 214ms/step - loss: 0.3789 - accuracy: 0.8676 - val_loss: 0.9471 - val_accuracy: 0.7046
```

```
Epoch 9/20
303/303 [==============================] - 65s 216ms/step - loss: 0.3167 - accuracy: 0.8885 - val_loss: 0.9635 - val_accuracy: 0.6972
303/303 [==============================] - 20s 60ms/step
76/76 [==============================] - 8s 73ms/step
Training model 2/3...
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
303/303 [==============================] - 110s 259ms/step - loss: 1.2622 - accuracy: 0.3363 - val_loss: 1.0973 - val_accuracy: 0.3366
Epoch 2/20
303/303 [==============================] - 67s 221ms/step - loss: 1.1890 - accuracy: 0.3581 - val_loss: 1.0391 - val_accuracy: 0.4571
Epoch 3/20
303/303 [==============================] - 66s 218ms/step - loss: 1.0323 - accuracy: 0.5142 - val_loss: 0.8397 - val_accuracy: 0.6324
Epoch 4/20
303/303 [==============================] - 66s 217ms/step - loss: 0.8517 - accuracy: 0.6433 - val_loss: 0.7801 - val_accuracy: 0.6683
Epoch 5/20
303/303 [==============================] - 65s 215ms/step - loss: 0.7133 - accuracy: 0.7192 - val_loss: 0.8132 - val_accuracy: 0.6790
Epoch 6/20
303/303 [==============================] - 65s 214ms/step - loss: 0.5864 - accuracy: 0.7735 - val_loss: 0.8025 - val_accuracy: 0.6939
Epoch 7/20
303/303 [==============================] - 65s 214ms/step - loss: 0.4790 - accuracy: 0.8248 - val_loss: 0.8860 - val_accuracy: 0.7001
Epoch 8/20
303/303 [==============================] - 65s 214ms/step - loss: 0.3730 - accuracy: 0.8694 - val_loss: 0.8574 - val_accuracy: 0.6951
Epoch 9/20
303/303 [==============================] - 66s 216ms/step - loss: 0.2983 - accuracy: 0.8963 - val_loss: 1.0039 - val_accuracy: 0.6951
303/303 [==============================] - 20s 60ms/step
76/76 [==============================] - 8s 73ms/step
Training model 3/3...
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
303/303 [==============================] - 110s 258ms/step - loss: 1.2716 - accuracy: 0.3407 - val_loss: 1.0966 - val_accuracy: 0.3367
```

```
# Ensemble prediction
def ensemble_predict(models, inputs):
    predictions = [model.predict(inputs) for model in models]
    avg_prediction = np.mean(predictions, axis=0)
    return np.argmax(avg_prediction, axis=1)

ensemble_preds = ensemble_predict(models, val_inputs)
accuracy = accuracy_score(y_test, ensemble_preds)
print(f"Ensemble Test Accuracy: {accuracy * 100:.2f}%")
```

```
76/76 [==============================] - 6s 72ms/step
76/76 [==============================] - 6s 73ms/step
76/76 [==============================] - 6s 72ms/step
Ensemble Test Accuracy: 70.50%
```

Ensemble Learning for Natural Language Inference with Stacking, Averaging, and L2 Regularization on XLM-RoBERTa with Bidirectional LSTM

```
X = df[['premise', 'hypothesis']].values
y = df['label'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
tokenizer = AutoTokenizer.from_pretrained("xlm-roberta-base")

def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs

train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

```
def create_model():
    base_model = TFAutoModelForSequenceClassification.from_pretrained("xlm-roberta-base", num_labels=3).roberta
    input_ids = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="input_ids")
    attention_mask = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="attention_mask")
    embeddings = base_model([input_ids, attention_mask])
    sequence_output = embeddings[0]

    # Adding a Bidirectional LSTM layer
    x = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True))(sequence_output)
    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    x = tf.keras.layers.Dropout(0.4)(x)
    x = tf.keras.layers.Dense(512, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001))(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    outputs = tf.keras.layers.Dense(3, activation='softmax')(x)

    model = tf.keras.Model(inputs=[input_ids, attention_mask], outputs=outputs)
    return model
```

```
NUM_MODELS = 3
models = []
stacked_features_train = []
stacked_features_val = []

for i in range(NUM_MODELS):
    print(f"Training model {i+1}/{NUM_MODELS}...")
    model = create_model()


    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6)
    lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-5 * 10**(epoch / 20))

    optimizer = keras.optimizers.Adam(learning_rate=1e-5)
    loss = keras.losses.SparseCategoricalCrossentropy()
    metric = keras.metrics.SparseCategoricalAccuracy('accuracy')
    model.compile(optimizer=optimizer, loss=loss, metrics=[metric])

    model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping, redu

    train_preds = model.predict(train_inputs)
    val_preds = model.predict(val_inputs)

    stacked_features_train.append(train_preds)
    stacked_features_val.append(val_preds)

    models.append(model)
    tf.keras.backend.clear_session()

# Stacking
stacked_features_train = np.concatenate(stacked_features_train, axis=1)
stacked_features_val = np.concatenate(stacked_features_val, axis=1)

stacker = keras.Sequential([
    keras.layers.Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(3, activation='softmax')
])
stacker.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
stacker.fit(stacked_features_train, y_train, validation_data=(stacked_features_val, y_test), epochs=50, batch_size=32)
```

```
Training model 1/3...
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
303/303 [==============================] - 115s 268ms/step - loss: 1.6617 - accuracy: 0.3340 - val_loss: 1.6395 - val_accuracy: 0.3424
Epoch 2/20
303/303 [==============================] - 71s 233ms/step - loss: 1.6446 - accuracy: 0.3401 - val_loss: 1.6311 - val_accuracy: 0.3412
Epoch 3/20
303/303 [==============================] - 70s 230ms/step - loss: 1.6232 - accuracy: 0.3741 - val_loss: 1.5675 - val_accuracy: 0.4686
Epoch 4/20
303/303 [==============================] - 69s 228ms/step - loss: 1.5549 - accuracy: 0.4593 - val_loss: 1.4884 - val_accuracy: 0.5149
Epoch 5/20
303/303 [==============================] - 69s 227ms/step - loss: 1.4270 - accuracy: 0.5834 - val_loss: 1.3372 - val_accuracy: 0.6452
Epoch 6/20
303/303 [==============================] - 68s 226ms/step - loss: 1.2929 - accuracy: 0.6816 - val_loss: 1.2604 - val_accuracy: 0.6848
Epoch 7/20
```

```
303/303 [==============================] - 69s 226ms/step - loss: 1.1547 - accuracy: 0.7558 - val_loss: 1.2416 - val_accuracy: 0.6939
Epoch 8/20
303/303 [==============================] - 68s 224ms/step - loss: 1.0389 - accuracy: 0.8099 - val_loss: 1.3455 - val_accuracy: 0.6774
Epoch 9/20
303/303 [==============================] - 68s 224ms/step - loss: 0.9448 - accuracy: 0.8407 - val_loss: 1.3433 - val_accuracy: 0.7013
Epoch 10/20
303/303 [==============================] - 68s 224ms/step - loss: 0.8361 - accuracy: 0.8844 - val_loss: 1.3821 - val_accuracy: 0.6749
Epoch 11/20
303/303 [==============================] - 68s 224ms/step - loss: 0.7826 - accuracy: 0.8977 - val_loss: 1.4390 - val_accuracy: 0.6894
Epoch 12/20
303/303 [==============================] - 69s 227ms/step - loss: 0.7254 - accuracy: 0.9164 - val_loss: 1.4969 - val_accuracy: 0.6922
303/303 [==============================] - 25s 64ms/step
76/76 [==============================] - 9s 76ms/step
Training model 2/3...
All PyTorch model weights were used when initializing TFXLMRobertaForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFXLMRobertaForSequenceClassification were not initialized from the PyTorch model and are
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
303/303 [==============================] - 115s 271ms/step - loss: 1.6593 - accuracy: 0.3363 - val_loss: 1.6364 - val_accuracy: 0.3457
Epoch 2/20
303/303 [==============================] - 71s 233ms/step - loss: 1.6460 - accuracy: 0.3354 - val_loss: 1.6328 - val_accuracy: 0.3511
Epoch 3/20
303/303 [==============================] - 69s 229ms/step - loss: 1.6470 - accuracy: 0.3391 - val_loss: 1.6238 - val_accuracy: 0.3300
Epoch 4/20
303/303 [==============================] - 69s 228ms/step - loss: 1.6313 - accuracy: 0.3345 - val_loss: 1.6164 - val_accuracy: 0.3300
Epoch 5/20
303/303 [==============================] - 69s 228ms/step - loss: 1.6175 - accuracy: 0.3350 - val_loss: 1.6071 - val_accuracy: 0.3511
Epoch 6/20
303/303 [==============================] - 69s 229ms/step - loss: 1.6077 - accuracy: 0.3295 - val_loss: 1.5970 - val_accuracy: 0.3556
Epoch 7/20
303/303 [==============================] - 69s 227ms/step - loss: 1.5949 - accuracy: 0.3366 - val_loss: 1.5851 - val_accuracy: 0.3511
Epoch 8/20
303/303 [==============================] - 69s 227ms/step - loss: 1.5823 - accuracy: 0.3373 - val_loss: 1.5715 - val_accuracy: 0.3511
Epoch 9/20
303/303 [==============================] - 69s 227ms/step - loss: 1.5667 - accuracy: 0.3353 - val_loss: 1.5565 - val_accuracy: 0.3511
Epoch 10/20
303/303 [==============================] - 69s 227ms/step - loss: 1.5506 - accuracy: 0.3329 - val_loss: 1.5388 - val_accuracy: 0.3511
Epoch 11/20
```

```
# Ensemble prediction using the stacker
stacked_preds = np.argmax(stacker.predict(stacked_features_val), axis=1)
accuracy = accuracy_score(y_test, stacked_preds)
print(f"Stacked Test Accuracy: {accuracy * 100:.2f}%")
```

```
76/76 [==============================] - 0s 1ms/step
Stacked Test Accuracy: 70.79%
```

```
# Ensemble prediction
def ensemble_predict(models, inputs):
    predictions = [model.predict(inputs) for model in models]
    avg_prediction = np.mean(predictions, axis=0)
    return np.argmax(avg_prediction, axis=1)

ensemble_preds = ensemble_predict(models, val_inputs)
accuracy = accuracy_score(y_test, ensemble_preds)
print(f"Ensemble Test Accuracy: {accuracy * 100:.2f}%")
```

```
76/76 [==============================] - 7s 76ms/step
76/76 [==============================] - 7s 76ms/step
76/76 [==============================] - 7s 76ms/step
Ensemble Test Accuracy: 70.87%
```

Natural Language Inference using Ensemble of XLM-RoBERTa Trained on SNLI, MNLI, ANLI, and XNLI Datasets with Stacking and Averaging - Final model considered for our prediction

```
X = df[['premise', 'hypothesis']].values
y = df['label'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
tokenizer = AutoTokenizer.from_pretrained("symanto/xlm-roberta-base-snli-mnli-anli-xnli")
```

```
def get_encodings(data):
    premises, hypotheses = data[:, 0], data[:, 1]
    encodings = tokenizer(premises.tolist(), hypotheses.tolist(), padding='max_length', truncation=True, max_length=128, return_tensors='tf')
    inputs = {key: tf.constant(val) for key, val in encodings.items()}
    return inputs

train_inputs = get_encodings(X_train)
val_inputs = get_encodings(X_test)
train_labels = tf.constant(y_train)
val_labels = tf.constant(y_test)
```

```
Downloading (…)okenizer_config.json: 100%                         398/398 [00:00<00:00, 37.2kB/s]

Downloading (…)tencepiece.bpe.model: 100%                         5.07M/5.07M [00:00<00:00, 85.1MB/s]

Downloading (…)/main/tokenizer.json: 100%                         9.08M/9.08M [00:00<00:00, 13.1MB/s]

Downloading (…)cial_tokens_map.json: 100%                         239/239 [00:00<00:00, 22.2kB/s]
```

```
def create_model():
    base_model = TFAutoModel.from_pretrained("symanto/xlm-roberta-base-snli-mnli-anli-xnli", from_pt=True)
    input_ids = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="input_ids")
    attention_mask = tf.keras.layers.Input(shape=(128,), dtype=tf.int32, name="attention_mask")
    embeddings = base_model([input_ids, attention_mask])
    pooled_output = embeddings[0][:, 0, :]

    x = tf.keras.layers.BatchNormalization()(pooled_output)
    x = tf.keras.layers.Dropout(0.3)(x)
    x = tf.keras.layers.Dense(512, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.3)(x)
    x = tf.keras.layers.Dense(256, activation='relu')(x)
    outputs = tf.keras.layers.Dense(3, activation='softmax')(x)

    model = tf.keras.Model(inputs=[input_ids, attention_mask], outputs=outputs)

    return model
```

```
NUM_MODELS = 3
models = []
stacked_features_train = []
stacked_features_val = []

for i in range(NUM_MODELS):
    print(f"Training model {i+1}/{NUM_MODELS}...")
    model = create_model()


    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6)
    lr_schedule = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-5 * 10**(epoch / 20))

    optimizer = keras.optimizers.Adam(learning_rate=1e-5)
    loss = keras.losses.SparseCategoricalCrossentropy()
    metric = keras.metrics.SparseCategoricalAccuracy('accuracy')
    model.compile(optimizer=optimizer, loss=loss, metrics=[metric])

    model.fit(train_inputs, train_labels, epochs=20, batch_size=32, validation_data=(val_inputs, val_labels), callbacks=[early_stopping, redu

    train_preds = model.predict(train_inputs)
    val_preds = model.predict(val_inputs)

    stacked_features_train.append(train_preds)
    stacked_features_val.append(val_preds)

    models.append(model)
    tf.keras.backend.clear_session()

# Stacking
stacked_features_train = np.concatenate(stacked_features_train, axis=1)
stacked_features_val = np.concatenate(stacked_features_val, axis=1)

stacker = keras.Sequential([
    keras.layers.Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(3, activation='softmax')
])
```

```
stacker.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
stacker.fit(stacked_features_train, y_train, validation_data=(stacked_features_val, y_test), epochs=50, batch_size=32)
```

```
Training model 1/3...
Downloading (…)lve/main/config.json: 100%                                   921/921 [00:00<00:00, 80.8kB/s]

Downloading pytorch_model.bin: 100%                                         1.11G/1.11G [00:02<00:00, 411MB/s]
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFXLMRobertaModel: ['classifier.dense.weight', 'class
- This IS expected if you are initializing TFXLMRobertaModel from a PyTorch model trained on another task or with another architecture (
- This IS NOT expected if you are initializing TFXLMRobertaModel from a PyTorch model that you expect to be exactly identical (e.g. init
Some weights or buffers of the TF 2.0 model TFXLMRobertaModel were not initialized from the PyTorch model and are newly initialized: ['r
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
303/303 [==============================] - 113s 257ms/step - loss: 0.4153 - accuracy: 0.8433 - val_loss: 0.2748 - val_accuracy: 0.8927 -
Epoch 2/20
303/303 [==============================] - 67s 221ms/step - loss: 0.2767 - accuracy: 0.8999 - val_loss: 0.2939 - val_accuracy: 0.8915 -
Epoch 3/20
303/303 [==============================] - 66s 218ms/step - loss: 0.2042 - accuracy: 0.9299 - val_loss: 0.3530 - val_accuracy: 0.8923 -
Epoch 4/20
303/303 [==============================] - 65s 215ms/step - loss: 0.1768 - accuracy: 0.9402 - val_loss: 0.3499 - val_accuracy: 0.8919 -
Epoch 5/20
303/303 [==============================] - 65s 216ms/step - loss: 0.1379 - accuracy: 0.9530 - val_loss: 0.4701 - val_accuracy: 0.8833 -
Epoch 6/20
303/303 [==============================] - 66s 217ms/step - loss: 0.1050 - accuracy: 0.9659 - val_loss: 0.4120 - val_accuracy: 0.8841 -
303/303 [==============================] - 20s 60ms/step
76/76 [==============================] - 9s 73ms/step
Training model 2/3...
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFXLMRobertaModel: ['classifier.dense.weight', 'class
- This IS expected if you are initializing TFXLMRobertaModel from a PyTorch model trained on another task or with another architecture (
- This IS NOT expected if you are initializing TFXLMRobertaModel from a PyTorch model that you expect to be exactly identical (e.g. init
Some weights or buffers of the TF 2.0 model TFXLMRobertaModel were not initialized from the PyTorch model and are newly initialized: ['r
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
303/303 [==============================] - 110s 259ms/step - loss: 0.3967 - accuracy: 0.8494 - val_loss: 0.2850 - val_accuracy: 0.8956 -
Epoch 2/20
303/303 [==============================] - 66s 219ms/step - loss: 0.2684 - accuracy: 0.9050 - val_loss: 0.3085 - val_accuracy: 0.8940 -
Epoch 3/20
303/303 [==============================] - 65s 216ms/step - loss: 0.2230 - accuracy: 0.9224 - val_loss: 0.3284 - val_accuracy: 0.8956 -
Epoch 4/20
303/303 [==============================] - 65s 216ms/step - loss: 0.1588 - accuracy: 0.9461 - val_loss: 0.3956 - val_accuracy: 0.8742 -
Epoch 5/20
303/303 [==============================] - 65s 215ms/step - loss: 0.1326 - accuracy: 0.9564 - val_loss: 0.4811 - val_accuracy: 0.8841 -
Epoch 6/20
303/303 [==============================] - 66s 217ms/step - loss: 0.1091 - accuracy: 0.9639 - val_loss: 0.5073 - val_accuracy: 0.8775 -
303/303 [==============================] - 21s 60ms/step
76/76 [==============================] - 9s 74ms/step
Training model 3/3...
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFXLMRobertaModel: ['classifier.dense.weight', 'class
- This IS expected if you are initializing TFXLMRobertaModel from a PyTorch model trained on another task or with another architecture (
- This IS NOT expected if you are initializing TFXLMRobertaModel from a PyTorch model that you expect to be exactly identical (e.g. init
Some weights or buffers of the TF 2.0 model TFXLMRobertaModel were not initialized from the PyTorch model and are newly initialized: ['r
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1/20
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
WARNING:tensorflow:Gradients do not exist for variables ['tfxlm_roberta_model/roberta/pooler/dense/kernel:0', 'tfxlm_roberta_model/rober
303/303 [==============================] - 109s 258ms/step - loss: 0.3900 - accuracy: 0.8525 - val_loss: 0.2767 - val_accuracy: 0.8944 -
Epoch 2/20
303/303 [==============================] - 67s 222ms/step - loss: 0.2745 - accuracy: 0.9032 - val_loss: 0.3089 - val_accuracy: 0.8948 -
Epoch 3/20
303/303 [==============================] - 65s 216ms/step - loss: 0.2058 - accuracy: 0.9281 - val_loss: 0.3420 - val_accuracy: 0.8903 -
Epoch 4/20
303/303 [==============================] - 65s 215ms/step - loss: 0.1735 - accuracy: 0.9395 - val_loss: 0.4212 - val_accuracy: 0.8812 -
Epoch 5/20
303/303 [==============================] - 65s 215ms/step - loss: 0.1249 - accuracy: 0.9578 - val_loss: 0.4872 - val_accuracy: 0.8767 -
Epoch 6/20
303/303 [==============================] - 66s 216ms/step - loss: 0.1068 - accuracy: 0.9674 - val_loss: 0.4956 - val_accuracy: 0.8795 -
303/303 [==============================] - 20s 60ms/step
76/76 [==============================] - 8s 73ms/step
Epoch 1/50
303/303 [==============================] - 2s 3ms/step - loss: 0.4284 - accuracy: 0.8976 - val_loss: 0.3226 - val_accuracy: 0.8993
Epoch 2/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2287 - accuracy: 0.9378 - val_loss: 0.3309 - val_accuracy: 0.9002
Epoch 3/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2241 - accuracy: 0.9360 - val_loss: 0.3298 - val_accuracy: 0.9002
Epoch 4/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2158 - accuracy: 0.9390 - val_loss: 0.3306 - val_accuracy: 0.9014
Epoch 5/50
Epoch 5/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2172 - accuracy: 0.9381 - val_loss: 0.3293 - val_accuracy: 0.9010
Epoch 6/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2172 - accuracy: 0.9374 - val_loss: 0.3256 - val_accuracy: 0.9002
Epoch 7/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2166 - accuracy: 0.9365 - val_loss: 0.3207 - val_accuracy: 0.9006
Epoch 8/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2127 - accuracy: 0.9387 - val_loss: 0.3224 - val_accuracy: 0.9018
Epoch 9/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2122 - accuracy: 0.9357 - val_loss: 0.3246 - val_accuracy: 0.8998
Epoch 10/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2141 - accuracy: 0.9363 - val_loss: 0.3243 - val_accuracy: 0.9010
Epoch 11/50
303/303 [==============================] - 1s 3ms/step - loss: 0.2100 - accuracy: 0.9384 - val_loss: 0.3246 - val_accuracy: 0.8985
```

```
# Ensemble prediction using the stacker
stacked_preds = np.argmax(stacker.predict(stacked_features_val), axis=1)
accuracy = accuracy_score(y_test, stacked_preds)
print(f"Stacked Test Accuracy: {accuracy * 100:.2f}%")
```

```
76/76 [==============================] - 0s 1ms/step
Stacked Test Accuracy: 90.06%
```

```
Epoch 16/50
```

```
# Ensemble prediction
def ensemble_predict(models, inputs):
    predictions = [model.predict(inputs) for model in models]
    avg_prediction = np.mean(predictions, axis=0)
    return np.argmax(avg_prediction, axis=1)

ensemble_preds = ensemble_predict(models, val_inputs)
accuracy = accuracy_score(y_test, ensemble_preds)
print(f"Ensemble Test Accuracy: {accuracy * 100:.2f}%")
```

```
76/76 [==============================] - 6s 75ms/step
76/76 [==============================] - 5s 72ms/step
76/76 [==============================] - 6s 74ms/step
Ensemble Test Accuracy: 89.77%
```

```
303/303 [==============================] - 1s 3ms/step - loss: 0.2049 - accuracy: 0.9381 - val_loss: 0.3156 - val_accuracy: 0.9010
```

So as the above model with stacker has the highest accuracy of 90.06%, we are considering that as our final model for prediction

```
Epoch 28/50
```

Predicting and Saving Test Results Using the Stacked Ensemble Model

```
303/303 [==============================] - 1s 3ms/step - loss: 0.2033 - accuracy: 0.9375 - val_loss: 0.3148 - val_accuracy: 0.8998
```

```
X_test_data = df_test[['premise', 'hypothesis']].values
test_inputs = get_encodings(X_test_data)

# Predicting using each base model
stacked_features_test = []
for model in models:
    test_preds = model.predict(test_inputs)
    stacked_features_test.append(test_preds)

# Stacking
stacked_features_test = np.concatenate(stacked_features_test, axis=1)

# Predicting using the above stacker model
stacked_test_preds = np.argmax(stacker.predict(stacked_features_test), axis=1)

df_test['predictions'] = stacked_test_preds
df_test[['premise', 'hypothesis', 'predictions']].to_csv('test_predictions.csv', index=False)
```

```
163/163 [==============================] - 12s 73ms/step
163/163 [==============================] - 12s 72ms/step
163/163 [==============================] - 12s 73ms/step
163/163 [==============================] - 0s 1ms/step
```

```
Epoch 43/50
```

```
df_test.head()
```